PowerShell Guru

Master of managing AD, Exchange, SharePoint and SQL

MichaelLappenbusch
IT-SPECIALIST APPLICATION DEVELOPMENT

Table of contents

1.Introduction to PowerShell 2
What is PowerShell? 2
Differences between PowerShell and command line 3
Introduction to PowerShell syntax 4
2.Basic Commands 5
Navigating the file system 5
View and edit files 7
managing processes 8
3.Advanced Commands 9
Manage user and group accounts 9
Manage packages and services 10
Manage network settings 11
4.PowerShell scripting 12
Introduction to the PowerShell scripting language 12
Creating Scripts 13
Automate tasks 14
5.MS Windows System Administration 15
Manage security settings 15
Manage storage 16
Monitoring and troubleshooting 17
6.Advanced Themes 18
Manage Active Directory 18
Manage Exchange 19
Managing SharePoint 20
Administering SQL Server 21
7.Conclusions 22
Summary of the most important commands 22
Tips and tricks for experienced users 23
imprint 25

1.Introduction to PowerShell

What is PowerShell?

PowerShell is a framework from Microsoft that allows administrators to run commands and scripts in a Windows environment. It is based on the .NET Framework and extends the capabilities of the batch scripting language that is included with Windows.

PowerShell provides a command-line interface (CLI) and scripting language that makes it possible to automatically perform tasks that would otherwise have to be performed manually. It also supports remote computer management and management of Windows services, Group Policy, Active Directory and much more.

An important aspect of PowerShell is the ability to interact with other applications and services using commands and scripts that support those applications and services. For example, PowerShell can be used to run commands to Microsoft Exchange to manage mailboxes and messages.

PowerShell also supports the creation of cmdlets, which are special commands that can be used with PowerShell. These cmdlets can be created by third parties or by Microsoft itself and extend the functionality of PowerShell.

PowerShell has been developed by Microsoft since 2006 and is regularly updated to include new features and improvements. It is available on all recent versions of Windows including Windows 7, Windows 8 and Windows 10.

Differences between PowerShell and command line

The command line, also known as "Command Prompt" or "cmd.exe", is a command line interface included with Windows that allows users to send commands directly to the operating system. In contrast, PowerShell is an advanced framework that builds on top of the command line and provides additional functionality and scripting capabilities.

An important difference between PowerShell and the command line is that PowerShell is based on the .NET Framework while the command line is based on the batch scripting language. This allows PowerShell to interact with other applications and services at a deeper level and perform more complex tasks.

PowerShell also offers a richer scripting language that allows tasks to be performed automatically and workflows to be automated. It also supports remote computer management and management of Windows services, Group Policy, Active Directory and much more.

Another difference is that PowerShell has a richer syntax that allows it to perform complex tasks and have more control over the operating system and managed resources. PowerShell also supports using cmdlets, which are special commands that can be used with PowerShell.

In summary, PowerShell is an enhanced version of the command line that provides additional functionality and scripting capabilities, allowing administrators to automatically perform tasks and automate workflows. It is a powerful framework that allows to go deeper into the operating system and perform more complex tasks.

Introduction to PowerShell syntax

PowerShell syntax is based on a combination of text commands and special operators that allow tasks to be performed and data to be processed. Some basic elements of PowerShell syntax are:

Commands: PowerShell commands are called cmdlets (short for "command-lets") and are special commands that can be used with PowerShell. Cmdlets have a specific syntax consisting of a verb and a noun, eg "Get-Process" (to display processes on a computer).

Operators: PowerShell supports different types of operators, such as arithmetic operators (+, -, *, /) or comparison operators (=, -lt, -gt). These operators make it possible to manipulate data and perform comparisons.

Variables: PowerShell supports the use of variables, which allow data to be cached and manipulated. Variables begin with a dollar sign ($) followed by a name, such as $variable.

Parameters: PowerShell cmdlets often support parameters that allow you to set specific properties or options of a command. Parameters are specified by a hyphen character (-) followed by the name of the parameter, such as "-Name" or "-Verbose".

Pipeline: PowerShell makes it possible to merge commands and use the output of one command as input for another command. This is called "pipelining" and allows more complex tasks to be performed and data to be processed by nesting commands within one another.

Scripts: PowerShell supports the creation of scripts that allow multiple commands to be run automatically. Scripts can be saved in a text file and can be run later by loading them into PowerShell.

There are many other aspects of PowerShell syntax, such as the use of functions and loops, that allow more complex tasks to be performed and data to be processed. There are also many different ways one can format and filter the output of commands to make the information you want more easily accessible.

One important thing to note is the case sensitivity in PowerShell. Although PowerShell is generally case insensitive, there are some exceptions, particularly with variables and parameters. It is therefore advisable to stick to the correct spelling to avoid mistakes.

There are also many resources available to help PowerShell users advance their knowledge and learn new skills. For example, Microsoft provides extensive documentation and guides that explain in

detail how to use the various aspects of PowerShell. There are also many online communities and forums that can help users solve problems and answer questions.

Overall, PowerShell is a powerful and versatile language that allows administrators to perform tasks and automate workflows automatically. With a thorough introduction to PowerShell syntax and regular practice and learning, users can improve their skills and work more effectively with the framework.

2.Basic Commands

Navigating the file system

Navigating the file system is one of the basic tasks that can be performed in PowerShell. PowerShell supports various commands and operators that make it possible to navigate through the file system and manage files and folders. Some important commands used to navigate the file system are:

Get-ChildItem (aka "dir" or "ls"): This command displays the subfolders and files in a specified path. The command can be used with various parameters to filter specific files or folders or to format the output.

Set-Location (aka "cd"): This command changes the current working directory. The command can be used with a path specification to change to a specific directory.

New-Item: This command creates a new folder or file in the current directory or in a specified path.

Remove-Item: This command deletes a specific folder or file from the file system. The command can be used with various parameters to filter specific files or folders.

Copy-Item: This command copies a specific folder or file to another directory.

Move-Item: This command moves a specific folder or file to another directory.

There are many other commands and operators that can be used to navigate the file system, such as:

Test-Path: This command checks whether a specific path exists in the file system and returns a boolean value.

Get-Item: This command returns information about a specific path in the file system, such as the full path, the size of the file, or the time it was created.

Get-Acl: This command returns the access control list (ACL) of a specific path in the file system.

Set-Acl: This command changes the access control list (ACL) of a specific path in the file system.

Compare-Object: This command compares two paths in the file system and returns the differences between them.

To navigate the file system, you can either specify the full path specification or use relative paths that refer to the current working path. You can also use wildcards to filter files or folders and use different parameters to format the output or show specific properties.

It is important to note that PowerShell commands and scripts that work with the file system can pose a high risk of data loss, especially if used incorrectly. Therefore, you should always make a backup of the data before running any commands or scripts, and it is recommended to run the commands in a test environment first.

View and edit files

In PowerShell, you can view and manipulate files using commands and operators specifically designed for this purpose. Some important commands used to view and edit files are:

Get-Content (aka "gc"): This command displays the contents of a specific file. The command can be used with various parameters to display specific lines or sections, or to format the output.

Set-Content (aka "sc"): This command replaces the content of a specific file or adds new content. The command can be used with a text string or an array of text lines.

Add-Content (aka "ac"): This command adds a new line or multiple lines of text to a specified file. The command can be used with a text string or an array of text lines.

Out-File: This command writes the output of another command to a specified file. The command can be used to write cmdlet output to a file instead of displaying it on the screen.

Select-String: This command searches for a specific string of characters in a specific file or folder and returns the lines where the string was found. The command can be used with various parameters to refine the search.

Get-ItemProperty: This command returns the properties of a specific file, such as the time it was created, the size of the file, or the permissions.

It is important to note that the commands and scripts that work with editing files can pose a high risk of data loss, especially if not used correctly. Therefore, you should always make a backup of the data before running any commands or scripts, and it is recommended to run the commands in a test environment first. It's also important that you check the permissions on the files before attempting to edit them to ensure you have the correct permissions to open and edit the files.

It is also important to be aware that some file editing commands cannot be undone and that unintentional modification or deletion of files can result in data loss. Therefore, it's wise to be cautious and ensure you know exactly what you're doing before executing any filesystem-modifying commands.

There are also many other tools and applications available to view and edit files that are integrated with PowerShell, such as the Windows Notepad applications such as Notepad or Microsoft Word and there are also third-party tools that one can use. However, if you need full control and automation, PowerShell is an excellent choice.

managing processes

In PowerShell, you can manage processes using commands and operators specifically designed for this purpose. Some important commands used to manage processes are:

Get-Process (aka "ps"): This command displays information about the running processes on a computer. The command can be used with various parameters to filter specific processes or to format the output.

Start-Process (aka "start"): This command starts a new process on a computer. The command can be used with a path specification to start a specific application.

Stop-Process (aka "stop"): This command terminates a specific process on a computer. The command can be used with the name or ID of the process.

Suspend-Process: This command pauses a specific process, which means the process stops running but keeps its state. The command can be used with the name or ID of the process.

Resume-Process: This command resumes a previously paused process. The command can be used with the name or ID of the process.

Wait-Process: This command blocks execution of the current script until a specific process finishes. The command can be used with the name or ID of the process.

Get-WmiObject: This command allows you to access WMI (Windows Management Instrumentation) objects, which contain information about processes and other system resources.

It is important to note that some of the commands used to manage processes can potentially have computer effects, especially if not used correctly. Before killing a process, you should make sure you understand the implications and that there is no alternative. It's also important to check permissions before attempting to start or kill processes to ensure you have the required permissions.

3.Advanced Commands

Manage user and group accounts

In PowerShell, you can manage user and group accounts using commands and operators specifically designed for this purpose. Some important commands used to manage user and group accounts are:

New-LocalUser: This command creates a new local user account on a computer. The command can be used to define the properties of the user account, such as the password.

Remove-LocalUser: This command deletes an existing local user account on a computer. The command can be used to permanently remove the user account and its data.

New-LocalGroup: This command creates a new local group on a computer. The command can be used to define the properties of the group, such as the group name.

Add-LocalGroupMember: This command adds a user or other group to an existing local group. The command can be used to manage user and group account memberships.

Get-LocalUser: This command returns information about a specific local user account on a computer, such as username, password, and group membership.

Get-LocalGroup: This command returns information about a specific local group on a computer, such as the group name and members.

It's important to note that these commands only manage local user and group accounts, not those in a domain.

It's also important that you check permissions before attempting to manage user and group accounts to ensure you have the required permissions. Some of the commands used to manage user and group accounts can potentially affect your computer, especially if used incorrectly. For example, deleting a user account may result in the loss of user data and settings.

It's also important to make sure you're using the right commands to perform the action you want. For example, there are commands like New-ADUser that can be used to create user accounts in an Active Directory domain, rather than New-LocalUser, which can only create local accounts.

It is also important to note that the security of user and group accounts is vital and that it is important to ensure passwords are secure and that unwanted users cannot gain access to the account. It is good practice to regularly review and manage user and group accounts to ensure only authorized individuals have access and that invalid accounts are removed.

Manage packages and services

In PowerShell, you can manage packages and services using commands and operators specifically designed for this purpose. Some important commands used to manage packages and services are:

Get-Service: This command displays information about the running services on a computer. The command can be used with various parameters to filter specific services or to format the output.

Start Service: This command starts a specific service on a computer. The command can be used with the name of the service.

Stop Service: This command stops a specific service on a computer. The command can be used with the name of the service.

Get-WindowsCapability: This command displays information about the available Windows features and components that can be installed or removed.

Add-WindowsCapability: This command adds a specific Windows feature or component to a computer. The command can be used with the name of the feature.

Remove-WindowsCapability: This command removes a specific Windows feature or component from a computer. The command can be used with the name of the feature.

It is important to note that some of the commands used to manage packages and services can potentially have computer effects, especially if used incorrectly. Before terminating a service or removing a feature, you should make sure you understand the implications and that there is no alternative. It's also important that you check permissions before attempting to install, remove, or manage any package or service to ensure you have the required permissions. It's also important to consider the dependencies of the different packages and services to ensure the correct packages and services are installed and removed.

It is also important to note that managing services and packages plays an important role in system administration and security. It is important to ensure that only needed services and packages are installed and that unwanted services and packages are removed to increase system security and optimize system performance. It is good practice to regularly check and manage the installed services and packages to ensure that the system is up to date and that there are no outdated or insecure components.

Manage network settings

In PowerShell, you can manage network settings using commands and operators specifically designed for this purpose. Some important commands used to manage network settings are:

Get-NetIPConfiguration: This command displays information about a computer's IP configuration, such as the IP address, subnet mask, and default gateway.

Set-NetIPInterface: This command allows you to change the properties of a network interface, such as the IP address or subnet mask.

Get-NetConnectionProfile: This command displays information about a computer's network connections, such as the connection name and connection status.

Set-DnsClientServerAddress: This command allows you to change the DNS server addresses for a computer.

Disable-NetAdapter: This command disables a specific network interface on a computer.

Enable-NetAdapter: This command enables a specific network interface on a computer.

Get-NetAdapter: This command returns information about a computer's network interfaces, such as name, MAC address, and status.

New-NetIPAddress: This command creates a new IP address for a specific network interface.

Remove-NetIPAddress: This command removes a specific IP address from a network interface.

It is important to note that some of the commands used to manage network settings can potentially affect the computer, especially if used incorrectly. Before making any change to the network settings, you should make sure that you understand the implications and that there is no alternative. It's also important that you check permissions before attempting to manage network settings to ensure you have the required permissions.

It is also important to note that managing network settings plays an important role in system administration and security. It is important to ensure that the network settings are configured correctly and that unwanted changes are avoided in order to increase the security of the system and to optimize the performance of the system. It is recommended to regularly check and manage the network settings to ensure that the system is working properly and that there are no configuration errors.

4.PowerShell scripting

Introduction to the PowerShell scripting language

PowerShell is a scripting language that makes it possible to perform automated tasks and simplify the administration of Windows systems. A PowerShell script is a text file that contains a sequence of PowerShell commands that run automatically when the file is run.

A PowerShell script typically begins by specifying the PowerShell interpreter, followed by the commands to run. For example:

```
#!/usr/bin/powershell

get service
```

In this simple example, the "Get-Service" command will run when the script is run and will display the information about the running services on the computer.

PowerShell scripts can also contain variables, loops, conditionals, and functions to extend script functionality. For example, a script can use a loop to create multiple user accounts and use variables to store the information needed for the accounts.

PowerShell scripts can be run in a number of ways, including double-clicking the script file, running the powershell.exe command and specifying the script path as an argument, or running the Invoke-Expression command and specifying the script path as an argument. It is also possible to automate PowerShell scripts as scheduled tasks or as part of workflows.

It is important to note that PowerShell scripts can potentially affect your computer, especially if they are not written correctly or if they are created by unauthorized people. It is therefore important to ensure that you use trusted sources for PowerShell scripts and that you check the scripts before running them to ensure that they are not dangerous. It's also important to check permissions before attempting to run PowerShell scripts to ensure you have the required permissions.

Creating Scripts

Creating PowerShell scripts requires a basic knowledge of PowerShell syntax and the use of PowerShell commands. To create a PowerShell script, you can use a text editor like Notepad to create a new text file and add the PowerShell commands you want.

An easy way to start creating PowerShell scripts is to record steps that are performed manually and convert those steps into PowerShell commands. It's also helpful to examine existing PowerShell scripts and understand how they work to get ideas for your own scripts.

It is important that every PowerShell script begins with what is called a "shebang" line, which indicates that it is a PowerShell script. For example:

```
#!/usr/bin/powershell
```

It is also important that each script is commented to explain how the script works and make it easier to understand and maintain. A good comment should describe the purpose of the script, the commands used, and the results.

When creating scripts, you should also consider security. It is important to ensure that the script does not inadvertently perform dangerous actions and that it cannot be executed by unauthorized people. To ensure this, you should ensure that the script can only be run with the required permissions and that it is validated before running it.

Automate tasks

Another important concept when creating PowerShell scripts is error handling. It is important to write the script in such a way that it detects possible errors and handles them appropriately. This can be achieved by using try-catch blocks and checking return values.

It is important to note that creating PowerShell scripts takes time and practice to develop the skills and knowledge to automate more complex tasks with PowerShell scripts. One of the key benefits of using PowerShell scripts is the ability to run tasks automatically instead of doing them manually.

An example of automating tasks is creating user accounts. Instead of creating each user account manually, a PowerShell script can be used to create multiple user accounts at once by running the appropriate commands automatically. Another example is automatically cleaning up temporary files on a computer. This can be done automatically by a PowerShell script instead of doing it manually.

PowerShell scripts can also be configured as scheduled tasks to run tasks automatically at specific times. This allows daily or weekly tasks like backing up data or updating software to be performed automatically without the need for manual input.

It is important to note that automating tasks may have computer effects and it is important to ensure that the automated tasks are configured correctly and that they do not have unexpected results. It's also important to check permissions before attempting to run automated tasks to ensure you have the required permissions.

5.MS Windows System Administration

Manage security settings

PowerShell offers several ways to manage security settings. You can use commands to monitor and manage the security of user accounts, groups, packages, and services.

An example of managing security settings is creating user accounts with strong passwords. You can create a PowerShell script that automatically creates user accounts and generates passwords that meet strong password requirements.

Another example is security event monitoring. You can use commands to read and analyze events from the event log to detect potential security issues.

You can also use commands to monitor the security of services and packages. For example, you can use commands to check what services are running on a computer and whether they are configured securely. You can also use commands to check the versions of installed packages and monitor for available security updates.

Another important concept in managing security settings is permissions management. You can use commands to review and manage user account, group, and resource permissions to ensure that only authorized users can access specific resources.

It is important to note that managing security settings is an ongoing process and it is important to regularly review and update security settings to ensure the system is secure. It's also important to check permissions before attempting to manage security settings to ensure you have the required permissions.

It's also important to note that it's important to carefully review and validate PowerShell scripts that manage security settings before you run them, to ensure they're not dangerous and that they produce the expected results. It's also good practice to log the execution of PowerShell scripts that manage security settings to track and troubleshoot issues.

There are also specialized PowerShell modules and tools like Microsoft's "Security Compliance Manager" (SCM) that are designed to manage and monitor security policies on Windows systems. These tools make it possible to implement and manage compliance policies in a simple and automated way.

Manage storage

PowerShell offers a wide range of commands and cmdlets that make it possible to manage storage resources on a Windows system.

Some of the most important cmdlets for managing Storage are:

Get-Disk: Displays information about all physical disks on the computer, including free and used space.

New-Partition: Creates a new partition on a disk.

Get-Volume: Displays information about all logical disks (volumes) on the computer.

Format Volume: Formats a volume with a specific file system.

Get-StoragePool: Displays information about all storage pools on the computer.

Get-VirtualDisk: Displays information about all virtual disks in a storage pool.

With these cmdlets one can perform a variety of tasks such as creating and deleting partitions, formatting volumes, creating and managing storage pools and virtual disks. It is also possible to use PowerShell to automate storage resources and create scripts that run periodically to simplify storage management.

There are also a number of other cmdlets that can be used to perform more specific tasks, such as managing RAID configurations, configuring storage replication, and managing storage classes.

It is important to note that some of the above cmdlets are only available on Windows Server systems and not on Windows Client systems.

Monitoring and troubleshooting

PowerShell provides a variety of commands and cmdlets that allow you to monitor and troubleshoot system and application performance and availability.

Some of the most important cmdlets for monitoring and troubleshooting are:

Get-EventLog: Displays events from the Windows event logs.

Get-Counter: Displays performance data of system resources.

Get-Service: Displays the status of services on the computer.

Get-Process: Displays running processes on the computer.

Get-WmiObject: Provides access to WMI objects to get information about system resources.

Test-Connection: Tests the connection to a remote computer.

These cmdlets can be used to perform a variety of tasks, such as monitoring events in the Windows event logs, querying system resource performance data, monitoring services and processes, querying information about WMI objects, and testing network connections .

It is also possible to create monitoring rules and policies using PowerShell to automatically monitor and report on specific events or performance data.

There are also a number of other cmdlets that can be used to perform more specific tasks, such as querying information about the security of the system or querying information about ongoing transactions in databases.

It is important to note that some of the above cmdlets are only available on Windows Server systems and not on Windows Client systems.

It's also important to make sure you choose the right tools and cmdlets to get the right information and make troubleshooting easier.

6.Advanced Themes

Manage Active Directory

Active Directory (AD) is a Microsoft service that allows administrators to manage users, computers, and other resources on a network. AD allows users and computers to be organized into domains, which in turn are organized into a hierarchy.

To manage AD with PowerShell, you must install the Active Directory Module for Windows PowerShell. Once that's done, you can use PowerShell cmdlets to perform various tasks such as create user accounts, add or remove computers, create and manage groups, and more.

Some examples of commonly used cmdlets to manage AD with PowerShell:

New-ADUser: Creates a new user account in AD

Set-ADUser: Changes the properties of an existing user account

Remove-ADUser: Deletes an existing user account

Get-ADUser: Retrieves information about an existing user account

New-ADGroup: Creates a new group in AD

Add-ADGroupMember: Adds members to an existing group

Get-ADGroup: Retrieves information about an existing group

There are many more cmdlets you can use to manage AD with PowerShell. It's a good idea to read the help for each cmdlet to make sure you're using it correctly and to understand what options are available.

Manage Exchange

Exchange is an email and calendaring system from Microsoft that allows businesses to manage email, calendar, contacts, and tasks. Exchange also offers features such as rules and policies management, email traffic monitoring, and mobile device support.

To manage Exchange with PowerShell, you must install the Exchange Module for Windows PowerShell. Once that's done, you can use PowerShell cmdlets to perform various tasks such as create user accounts, manage email addresses, manage mailboxes, create rules, and more.

Some examples of commonly used cmdlets to manage Exchange with PowerShell:

New-Mailbox: Creates a new mailbox

Set-Mailbox: Changes the properties of an existing mailbox

Remove-Mailbox: Deletes an existing mailbox

Get-Mailbox: Retrieves information about an existing mailbox

New-MailboxPermission: Grants permissions on a mailbox

New-TransportRule: Creates a new transport rule

Get-TransportRule: Retrieves information about an existing transport rule

There are many other cmdlets you can use to manage Exchange with PowerShell. It's a good idea to read the help for each cmdlet to make sure you're using it correctly and to understand what options are available. There are also some special cmdlets that are only available in Exchange Online or Exchange Server, so it's a good idea to refer to the specific version.

Managing SharePoint

SharePoint is a Microsoft platform that enables companies to manage and share documents, web content, applications and processes. SharePoint also provides features such as real-time collaboration, access rights management, content search, and custom solution development.

To manage SharePoint with PowerShell, you must install the SharePoint Module for Windows PowerShell. Once that's done, you can use PowerShell cmdlets to perform various tasks such as creating websites, managing lists and libraries, managing user access rights, managing content structures, and more.

Some examples of commonly used cmdlets to manage SharePoint with PowerShell:

New-SPWeb: Creates a new website

Set-SPWeb: Changes the properties of an existing website

Remove-SPWeb: Deletes an existing website

Get-SPWeb: Retrieves information about an existing website

New-SPList: Creates a new list

Add-SPListItem: Adds a new item to an existing list

Get-SPList: Retrieves information about an existing list

Grant-SPObjectSecurity: Grants access rights to a SharePoint object

Get-SPUser: Retrieves information about an existing user in SharePoint

There are many other cmdlets you can use to manage SharePoint with PowerShell. It's a good idea to read the help for each cmdlet to make sure you're using it correctly and to understand what options are available. It's also a good idea to familiarize yourself with the basic concepts of SharePoint before you start managing it with PowerShell to make the commands easier to understand.

Administering SQL Server

SQL Server is a relational database management system (RDBMS) from Microsoft that enables companies to store, query and manage data. SQL Server also provides features such as security management, business intelligence (BI) support, and custom solution development.

To manage SQL Server with PowerShell, you must install the SQL Server Module for Windows PowerShell. Once this is done, you can use PowerShell cmdlets to perform various tasks such as creating databases, managing tables, managing user access rights, managing security, and more.

Some examples of commonly used cmdlets to manage SQL Server with PowerShell:

Invoke-Sqlcmd: Runs a Transact-SQL query or script on a SQL Server instance.

Backup-SqlDatabase: Creates a backup of a SQL Server database

Restore-SqlDatabase: Restores a SQL Server database from a backup

New-SqlLogin: Creates a new SQL Server login object

Remove-SqlLogin: Deletes an existing login item

Add-SqlAvailabilityDatabase: Adds a database to an availability group

Get-SqlInstance: Retrieves information about a SQL Server instance

There are many other cmdlets you can use to manage SQL Server with PowerShell. It's a good idea to read the help for each cmdlet to make sure you're using it correctly and to understand what options are available. It's also a good idea to familiarize yourself with the basic concepts of SQL Server before you start managing it with PowerShell to make the commands easier to understand.

7.Conclusions

Summary of the most important commands

Active Directory:

New-ADUser: Creates a new user account in AD

Set-ADUser: Changes the properties of an existing user account

Remove-ADUser: Deletes an existing user account

Get-ADUser: Retrieves information about an existing user account

New-ADGroup: Creates a new group in AD

Add-ADGroupMember: Adds members to an existing group

Get-ADGroup: Retrieves information about an existing group

Exchange:

New-Mailbox: Creates a new mailbox

Set-Mailbox: Changes the properties of an existing mailbox

Remove-Mailbox: Deletes an existing mailbox

Get-Mailbox: Retrieves information about an existing mailbox

New-MailboxPermission: Grants permissions on a mailbox

New-TransportRule: Creates a new transport rule

Get-TransportRule: Retrieves information about an existing transport rule

SharePoint:

New-SPWeb: Creates a new website

Set-SPWeb: Changes the properties of an existing website

Remove-SPWeb: Deletes an existing website

Get-SPWeb: Retrieves information about an existing website

New-SPList: Creates a new list

Add-SPListItem: Adds a new item to an existing list

Get-SPList: Retrieves information about an existing list

Grant-SPObjectSecurity: Grants access rights to a SharePoint object

Get-SPUser: Retrieves information about an existing user in SharePoint

SQL Server:

Invoke-Sqlcmd: Runs a Transact-SQL query or script on a SQL Server instance

Backup-SqlDatabase: Creates a backup of a SQL Server database

Restore-SqlDatabase: Restores a SQL Server database from a backup

New-SqlLogin: Creates a new SQL Server login object

Remove-SqlLogin: Deletes an existing login item

Add-SqlAvailabilityDatabase: Adds a database to an availability group

Get-SqlInstance: Retrieves information about a SQL Server instance

There are many other cmdlets that provide commands and options that can be used as needed to automate and streamline Active Directory, Exchange, SharePoint, and SQL Server administration. However, it's important to note that managing these systems with PowerShell requires you to have in-depth knowledge and experience of the specific systems and the PowerShell language. Using the commands incorrectly can lead to unexpected results or even loss of data. It is therefore recommended that you familiarize yourself with the documentation and technical support before using the commands.

Tips and tricks for experienced users

can improve the efficiency and accuracy of managing Active Directory, Exchange, SharePoint and SQL Server:

Use aliases: PowerShell has many cmdlets with long and complex names. You can use aliases to invoke these commands more easily and quickly.

Use tab completion: Pressing the tab key is a quick way to invoke a command's available options instead of typing them in manually.

Use compound cmdlet commands: PowerShell provides compound cmdlet commands that make it possible to run multiple commands on a single line. For example, you can use the command "Get-ADUser -Filter * | Export-CSV -Path C:\Users.csv" to export all users from AD to a CSV file.

Create scripts: Instead of typing each command manually, you can create scripts to automatically perform repetitive tasks.

Use the pipeline: The pipeline allows the output of one command to be used as input for the next command, making it easier to automate tasks and improve code readability.

Use the parameter table syntax: You can specify multiple values for a given parameter at once by using a table of values. For example, you can use the command "Add-ADGroupMember -Identity "Marketing" -Members @("User1","User2","User3")" to add multiple users to a group at once.

Use the -WhatIf option: This option shows what changes will be made before actually running the command, which helps prevent unintended effects.

Use -confirm option: With this option, the user will be asked if he wants to confirm the execution of the command before executing it.

Use the -ErrorAction option: This option makes it possible to control how PowerShell should react to errors.

Use the -Verbose option: This option prints detailed information about the execution of the command, which helps to identify and solve problems faster.

It's important to note that using these tips and tricks requires a combination of a deep understanding of the PowerShell language and the underlying systems you're managing for. It's also important to always be aware of the impact of changes and make regular backups to avoid data loss. It's also a good idea to leverage documentation and technical support to ensure you're getting the best management possible.

imprint

This book was published under the
Creative Commons Attribution-NonCommercial-NoDerivatives (CC BY-NC-ND) licensereleased.

This license allows others to use and share the book for free as long as they credit the author and source of the book and do not use it for commercial purposes.

Author: Michael Lappenbusch

E-mail:admin@perplex.click

home page:https://www.perplex.click

Release year: 2023

Some of the content comes from:ChatGPT